université PARIS-SACLAY

Introduction à la programmation fonctionnelle
L2 Informatique

semaine du 7 novembre 2020

TP n° 5

Consignes les exercices ou questions marqués d'un ⋆ devront être rédigés sur papier (afin de se préparer aux épreuves écrites de l'examen). En particulier, il est recommandé d'être dans les même conditions qu'en examen : pas de document ni de calculatrice. Tous les TPs se font sous Linux.

0 Prise en main de l'environnement

Pour faciliter les TPs en mode distanciel, nous utiliseront le serveur Jupiter hébergé par le LAL (Laboartoire Accélérateur Linéaire). Dans cet environnement, vous avez accès, via un navigateur Web à :

— un espace de stockage
— un terminal Unix
— un éditeur de texte avec la coloration syntaxique pour OCaml
— les commandes vues en cours : `ocamlc`, `ocaml`, ...

1. Se rendre sur `https://jupytercloud.lal.in2p3.fr/`
2. Cliquer sur se connecter
3. Utiliser l'authentification centralisée de Paris-Saclay (si vous êtes déjà connecté via eCampus, il est possible que l'authentification se fasse automatiquement)
4. Si vous êtes redirigé vers la page de garde : `https://jupytercloud.lal.in2p3.fr/`, c'est un bug connu. Effacez vos cookies ou ouvrez l'URL dans une fenêtre de navigation privée
5. Arrivé sur l'interface, choisissez « Start My Server » (bouton vert)

Vous vous retrouvez dans un navigateur de fichier. Vous pouvez créer des répertoires et des fichiers. Le bouton « New » à droite comporte plusieurs options dont :

— Create Terminal : ouvre un terminal dans un autre onglet de votre navigateur
— Create Texte File : ouvre un éditeur de texte

Dans l'éditeur de texte, si vous renommez le fichier en lui donnant l'extension « `.ml` » la coloration syntaxique pour OCaml est activée.

Il est **vivement recommandé** de télécharger vos fichiers après chaque exercice, pour en avoir une copie sur votre ordinateur personnel. Les sauvegardes sur cet espace de travail expérimental ne sont pas garanties, et c'est un espace distinct de vos répertoires personnels sur les machines de l'Université.

1 Tris en folie

Le but de l'exercice est d'implémenter deux fonctions de tri sur les listes et d'en évaluer les performances. On évaluera aussi les performances de la fonction prédéfinie `List.sort` vue en cours.

1. Écrire une fonction `rand_list : int -> int list` qui prend en argument un entier n et qui construit une liste de taille n contenant des entiers aléatoires compris entre 0 et $n-1$ inclus. On pourra utiliser la fonction `Random.int n` qui renvoie un entier dans cet intervalle.
2. Écrire une fonction `time : ('a -> 'b) -> 'a -> 'b * float`. `time f x` calcule le temps mis pour exécuter `f x` et renvoie un couple formé du résultat et du temps d'exécution en millisecondes. On rappelle que la fonction `Unix.gettimeofday()` renvoie un flottant représentant l'heure qu'il est, exprimée en secondes écoulée depuis le 1er janvier 1970, 00 :00 :00 UTC (cette date est appelée *Epoch*).
   Attention : à partir de cette question, il faut utiliser la ligne de commande suivante pour compiler votre programme :

   ```
   ocamlopt -o test.exe unix.cmxa test.ml
   ```

où `test.ml` est le nom de votre programme OCaml.

3. Écrire une fonction `insert_sort : ('a -> 'a -> int) -> 'a list -> 'a list` qui prend en argument une fonction de comparaison (cf. `List.sort`) et tri la liste passée en argument en utilisant l'algorithme du tri par insertion.
   — définir dans un premier temps une fonction imbriquée récursive `ins : 'a list -> 'a -> 'a list` qui prend en argument une liste et un élément et va insérer l'élément à la bonne position dans la list (en utilisant la fonction de comparaison).
   — une fois la fonction `ins` définie, recréer une liste triée en insérant à la bonne position tous les éléments de la liste initiale (on fera une utilisation judicieuse de `List.fold_lfet`).
4. Écrire une fonction `test_sort : (int list -> int list) -> string -> int list -> unit`. `test_sort : f n sizes` :
   — attend en argument une fonction `f` qui trie une liste d'entiers par ordre croissant
   — attend en argument une chaîne de caractères `n`
   — attend en argument une liste d'entiers `sizes`
   — Commence par appeler `Random.init 42`
   — Affiche sur une ligne la chaîne `n`
   — Et pour chaque entier de la liste `sizes`, génère une liste aléatoire de cette taille, calcule le temps mis pour la trier et affiche la taille de la liste et ce temps dans la console.
   — Enfin la fonction affiche deux retours à la ligne.

   Par exemple, si dans le code OCaml on appelle :

```
1  test_sort (insert_sort compare) "insert_sort" [0;1;10;20;50;100] ;;
```

   on obtient l'affichage

```
insert_sort
0 0.000000
1 0.000000
10 0.000954
20 0.006199
50 0.026941
100 0.087976
```

5. Tester votre tri par insertion et déterminer à partir de quelle taille de liste on dépasse 16ms [1].
6. Écrire maintenant la fonction `quick_sort` vue en cours. La tester et déterminer de la même façon la taille des listes que l'on peut trier en moins de 16ms.
7. Tester la fonction `List.sort` et la comparer à `quick_sort` et `insert_sort`.

1. C'est le temps de calcul auquel on a droit entre deux *frames* d'un jeu vidéo fluide affichant 60 images par secondes : $\frac{1000}{60} = 16.666$. Si le calcul (collisions, logique du jeu, points de vie, ...) prend plus de temps, le jeu laggue.